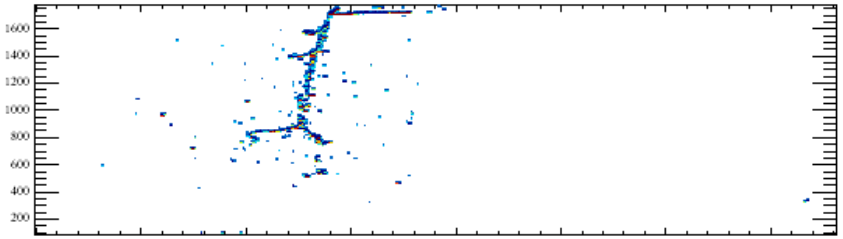
Update on fuzzy clustering

Ben Carls

Since the last update

- Concentrating on evaluations, will demonstrate efficiency and purity
- Also looking into using hit charge (see Bruce Baller's talk: https://cdcvs.fnal.gov/redmine/documents/591), that is not ready for today however

Overview of the algorithm



Run fuzzy clustering

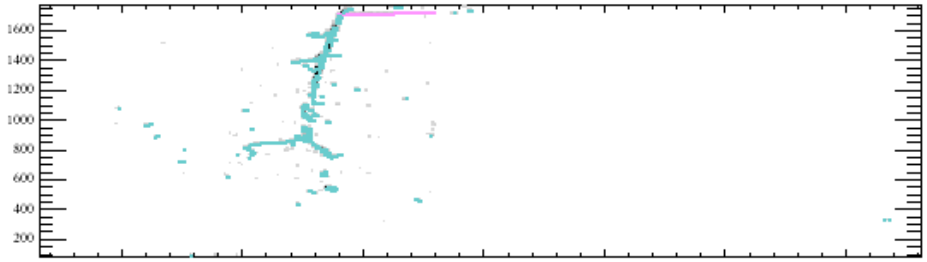
Check Xie-Beni index, determine number of clusters

Merge nearby fuzzy clusters

Run Hough line finder

Merge Hough lines

Merge fuzzy cluster remnants into merged Hough lines



Purity and Efficiency

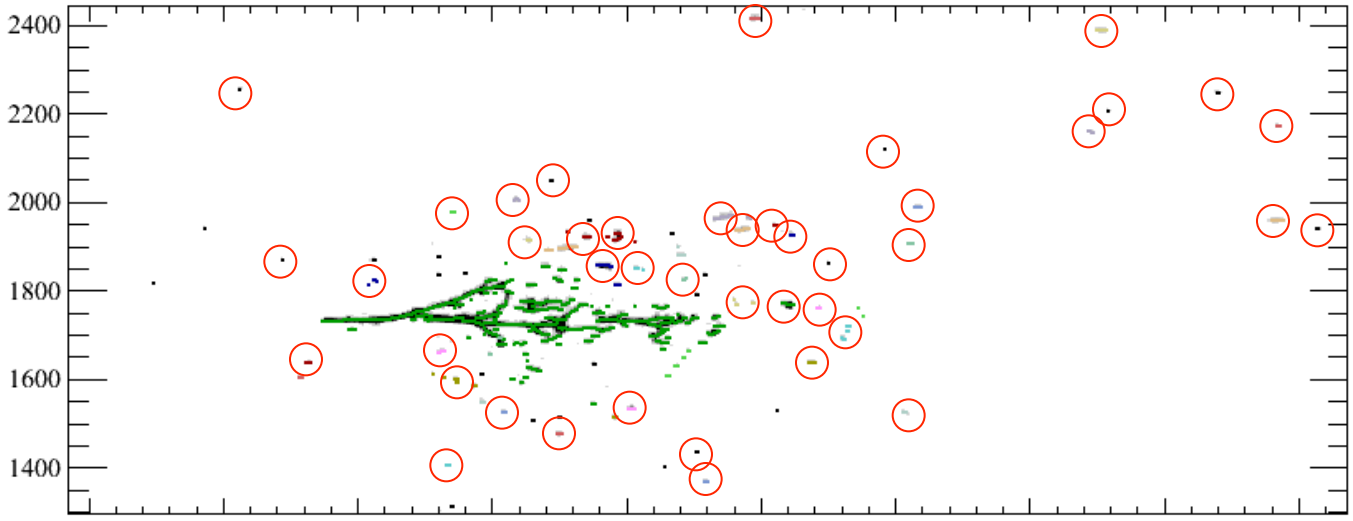
- Evaluated using the efficiency and purity found in the BackTracker

$$\text{Purity} = \frac{\#\,\text{hits from trackID in cluster}}{\text{total \# hits in cluster examined}}$$

$$\text{Efficiency} = \frac{\#\,\text{hits from trackID in cluster}}{\text{total \# hits for that trackID}}$$

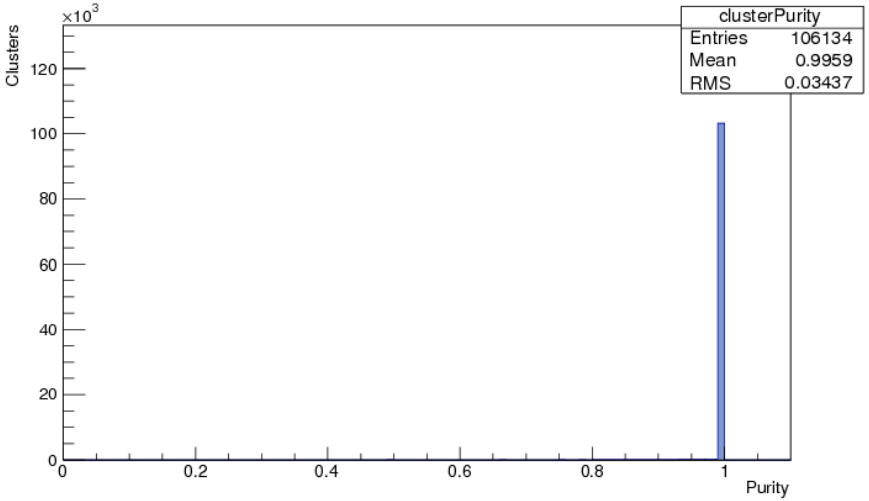
  Select values for trackIDs having the highest purity
- Comparison is tricky since Fuzzy Clustering and DBSCAN/Hough/Line Merger give different products
- Looked at 1,000 CCQE events

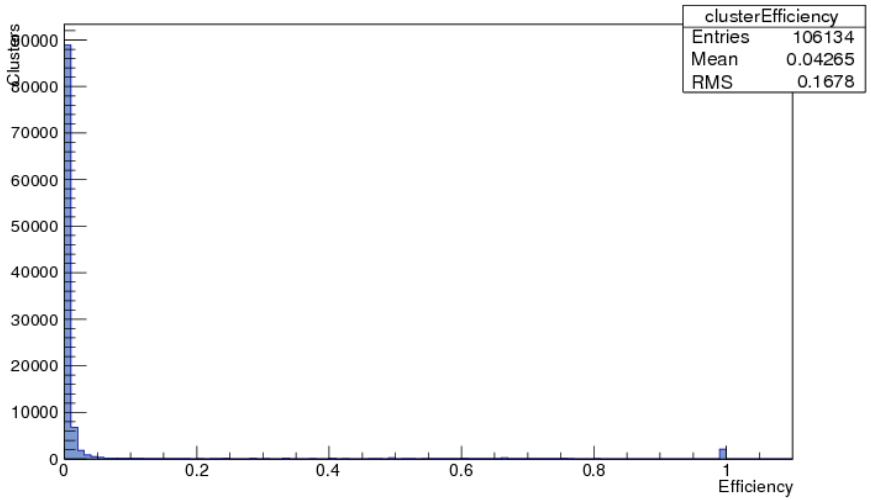
An example with DBSCAN



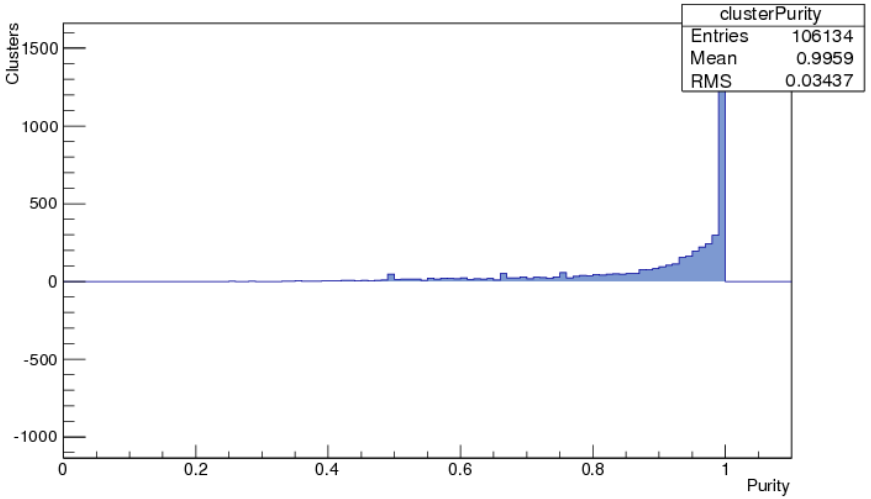
- We expect a lot of high purity clusters, lots of little clusters with only the hits from the electron shower
- We expect a lot of low efficiency clusters though, lots of the little clusters will have only a small portion of hits from the electron shower
- I've highlighted several of these in red

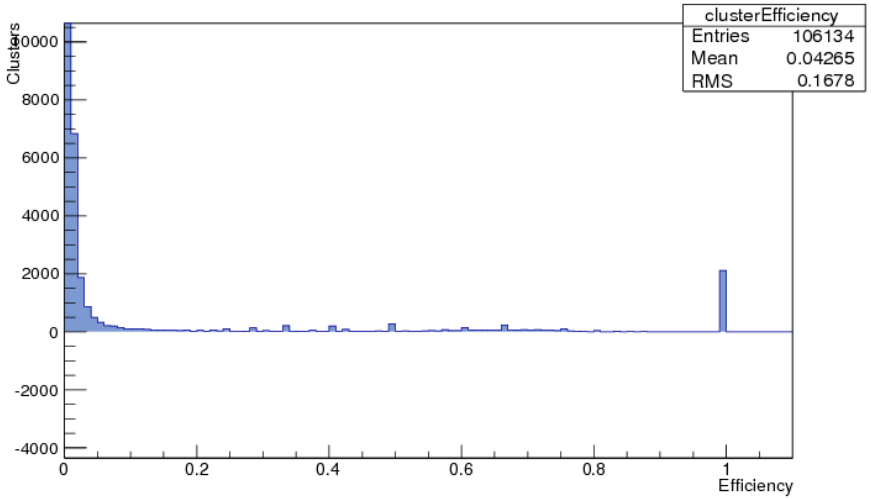
DBSCAN


clusterPurity
Entries 106134
Mean 0.9959
RMS 0.03437
Clusters
×10^3
120
100
80
60
40
20
0
0
0.2
0.4
0.6
0.8
1
Purity


clusterEfficiency
Entries 106134
Mean 0.04265
RMS 0.1678
Clusters
80000
70000
60000
50000
40000
30000
20000
10000
0
0
0.2
0.4
0.6
0.8
1
Efficiency

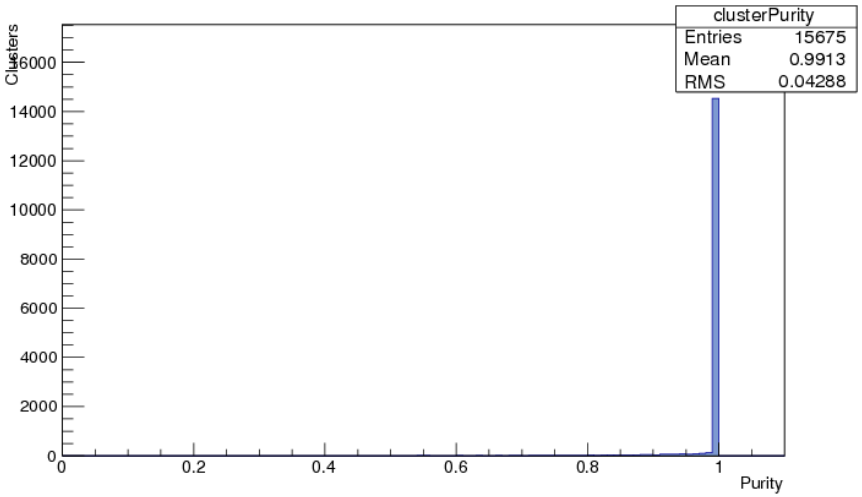
DBSCAN

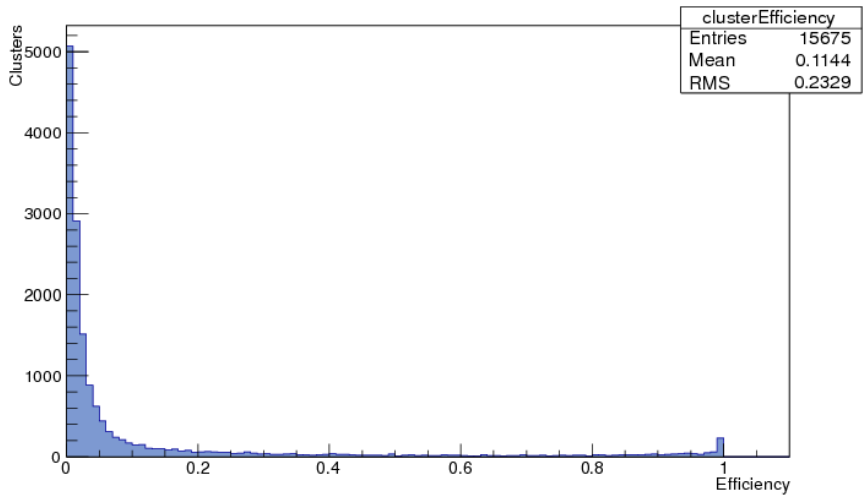




Zoomed in

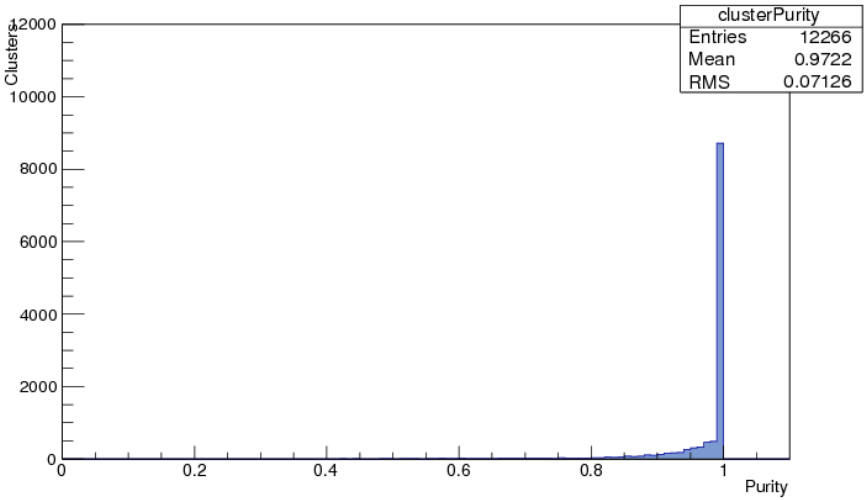
Line Merger

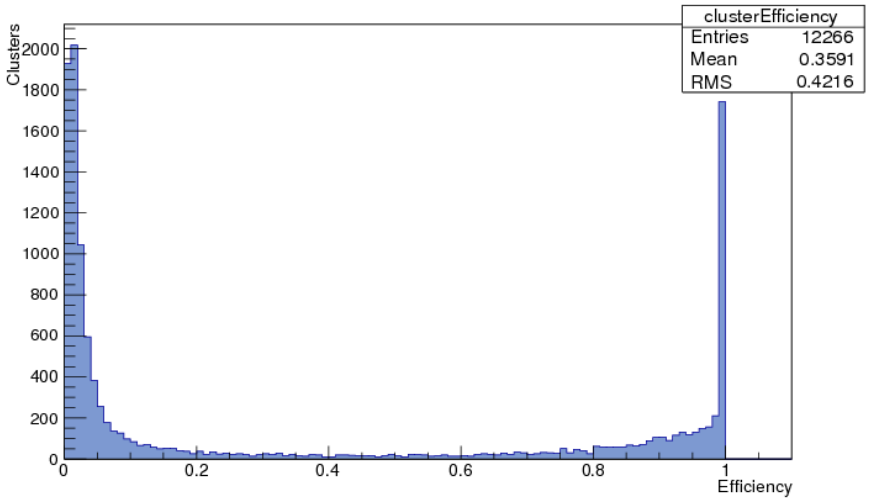




This comes from running DBSCAN, Hough Line Finder, then Line Merger

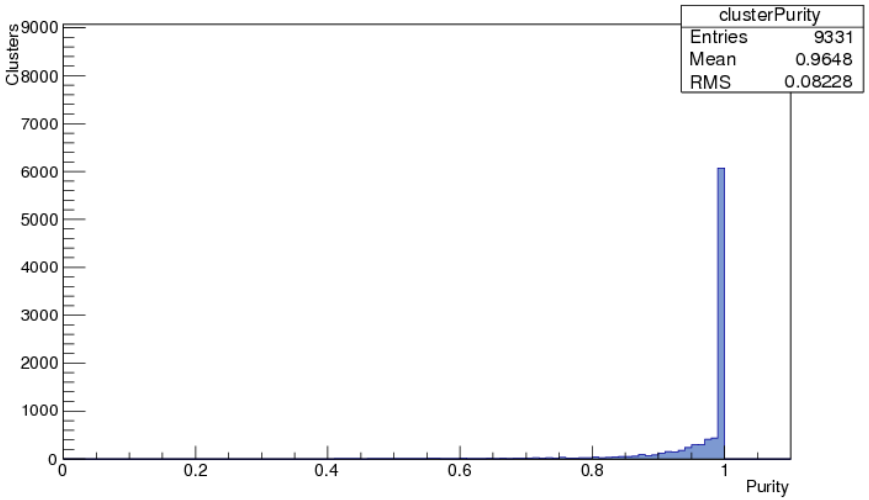
Fuzzy Clustering

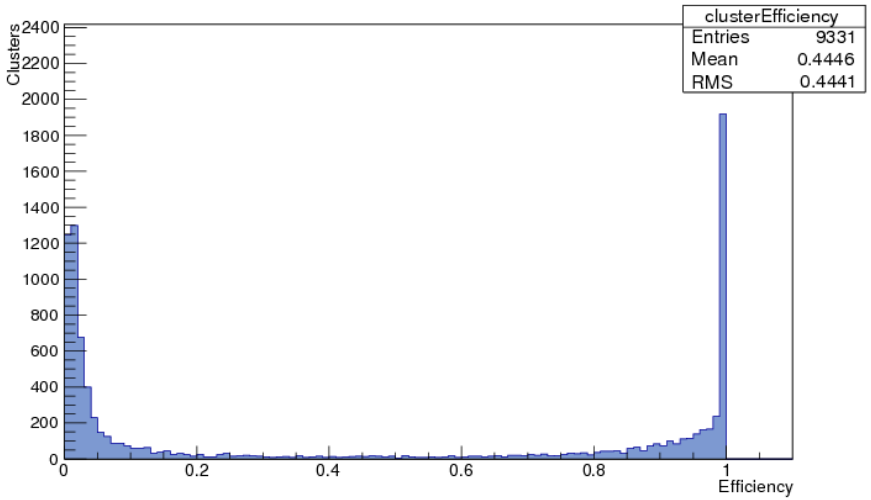




Out of the box, using 15° merge between lines

Fuzzy Clustering



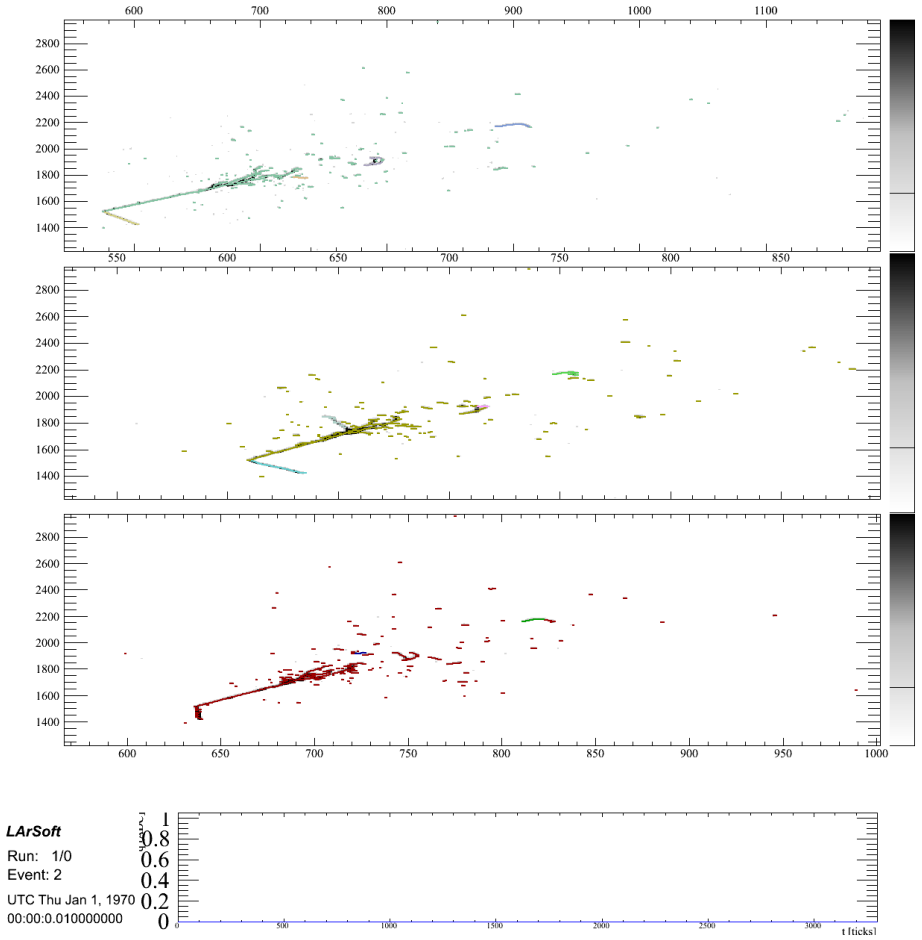


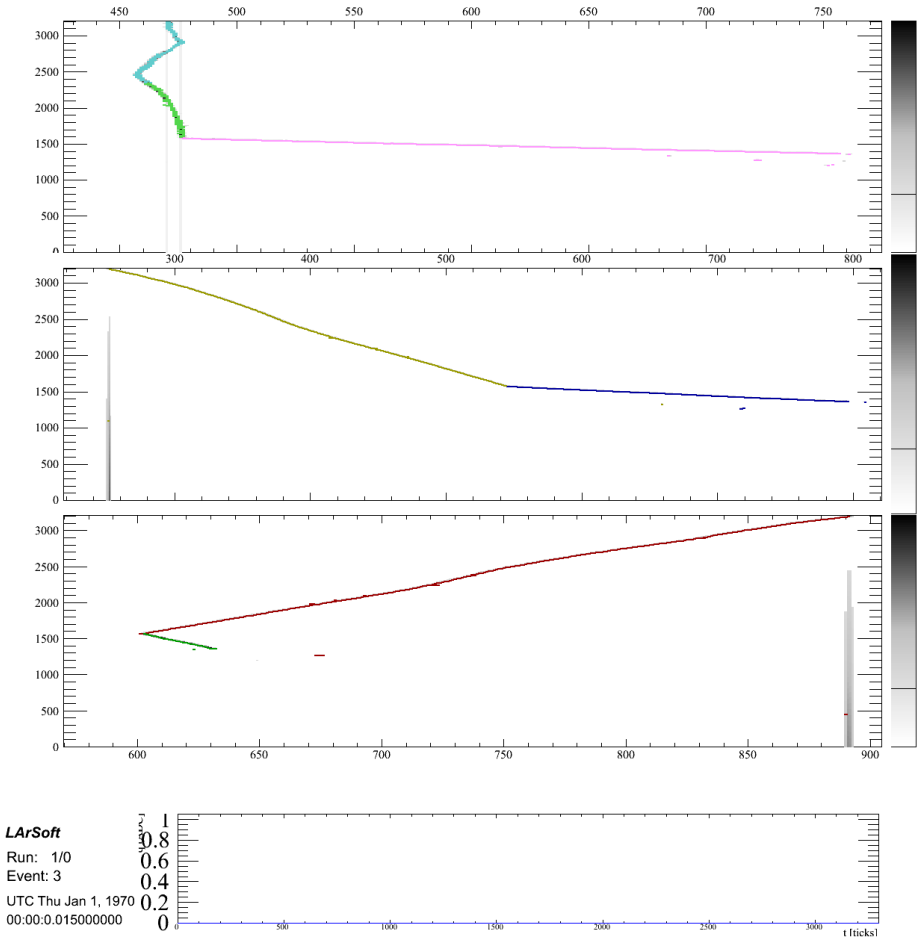
Out of the box, using 30° merge between lines

Where it stands now

- I want to continue looking at how different parameters alter purity and efficiency
- Look for more comparisons and optimizations, very open to suggestions
- Complete documentation, draft is on Redmine

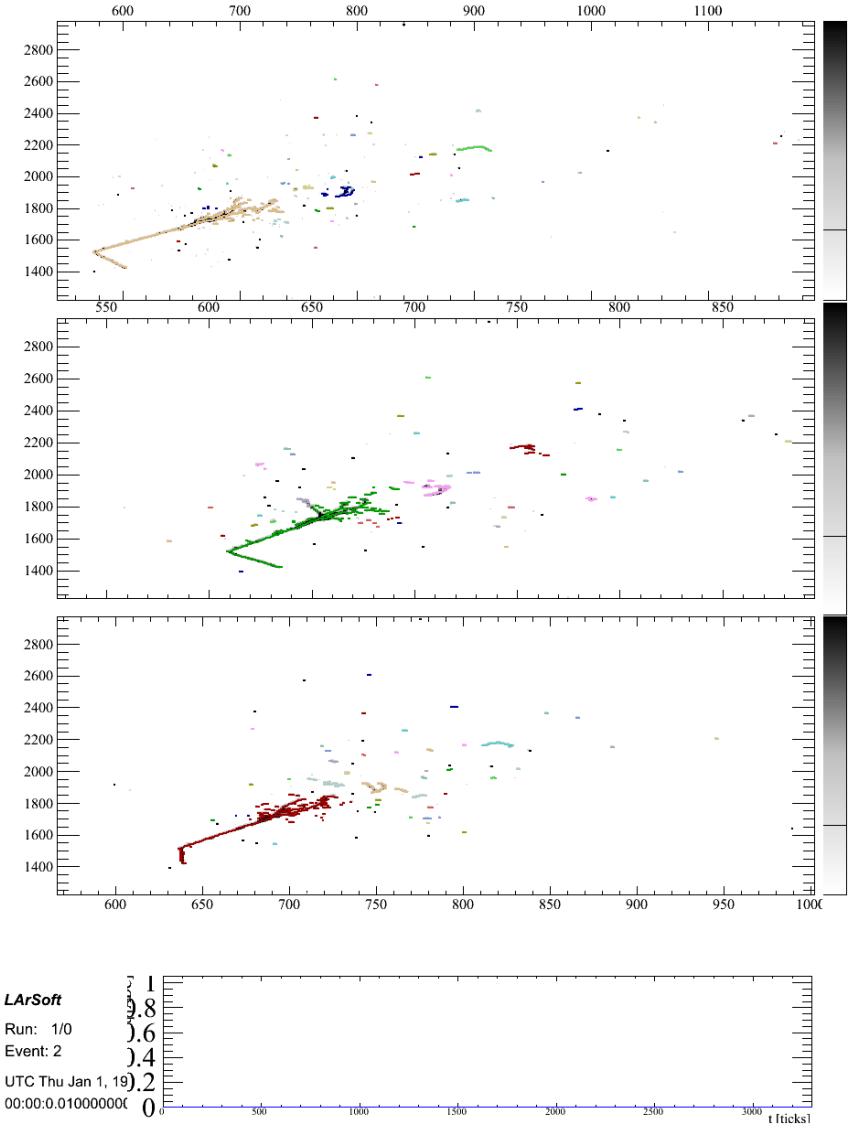
Fuzzy Clustering

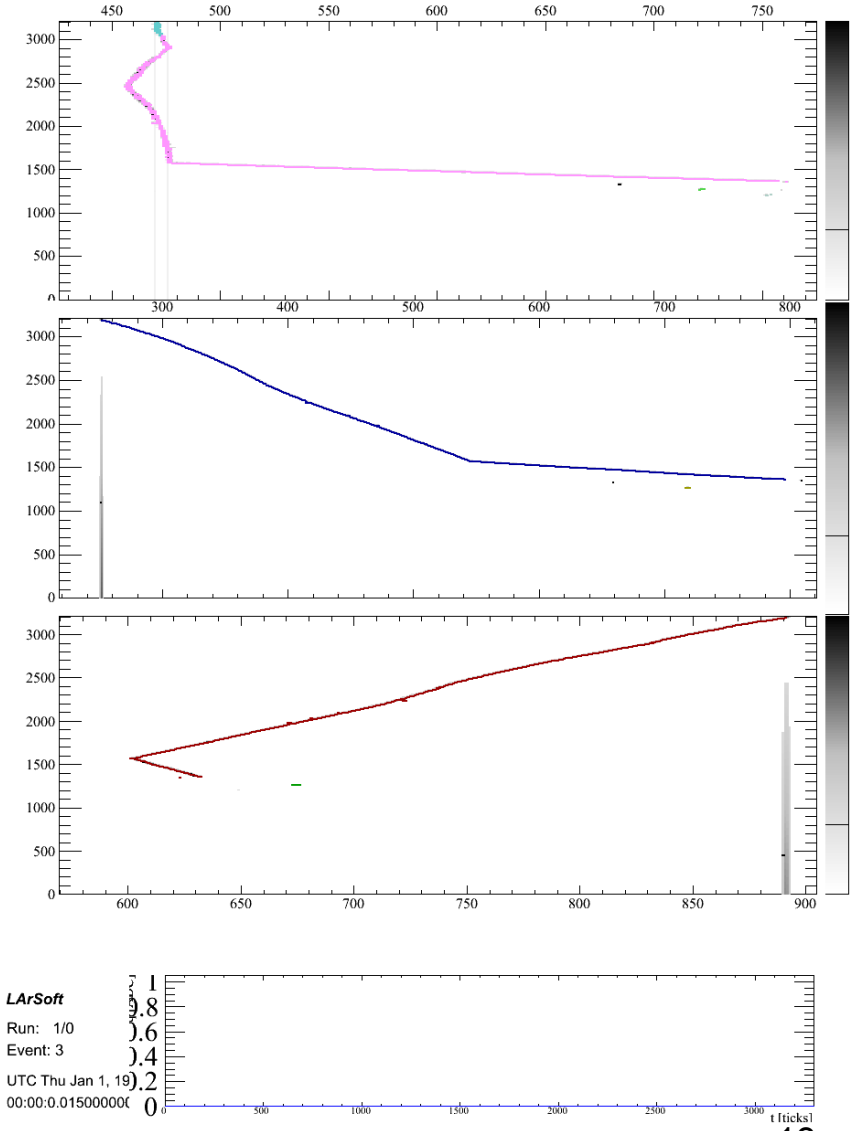




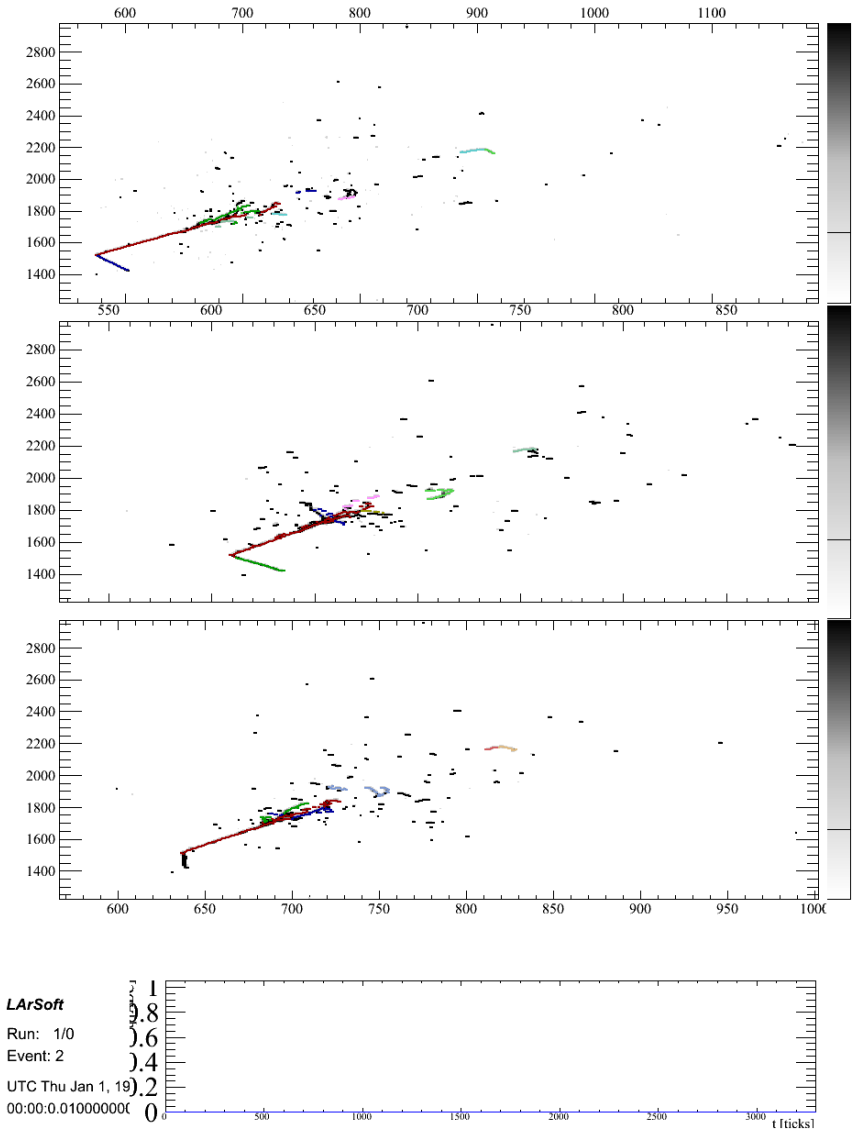
Out of the box, using 30° merge between lines

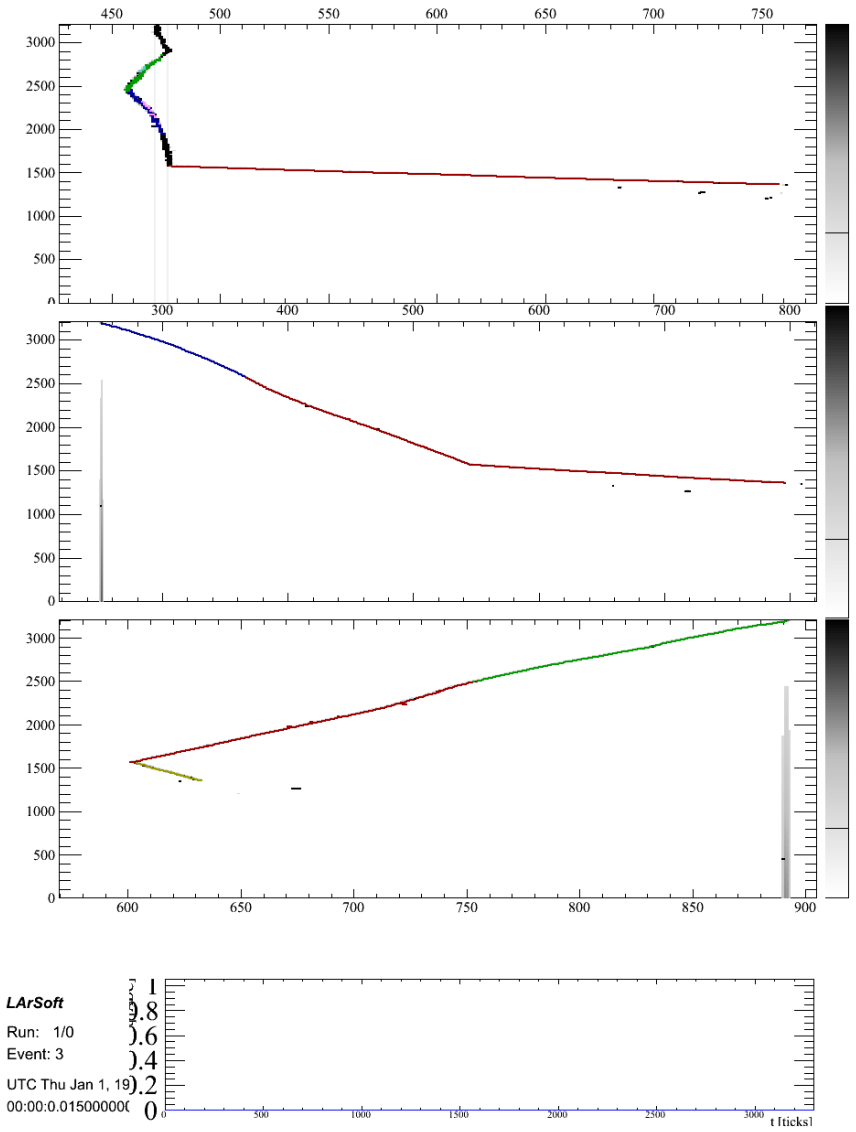
DBSCAN


LArSoft
Run: 1/0
Event: 2
UTC Thu Jan 1, 19
00:00:0.01000000(
t [ticks]


LArSoft
Run: 1/0
Event: 3
UTC Thu Jan 1, 19
00:00:0.01500000(
t [ticks]

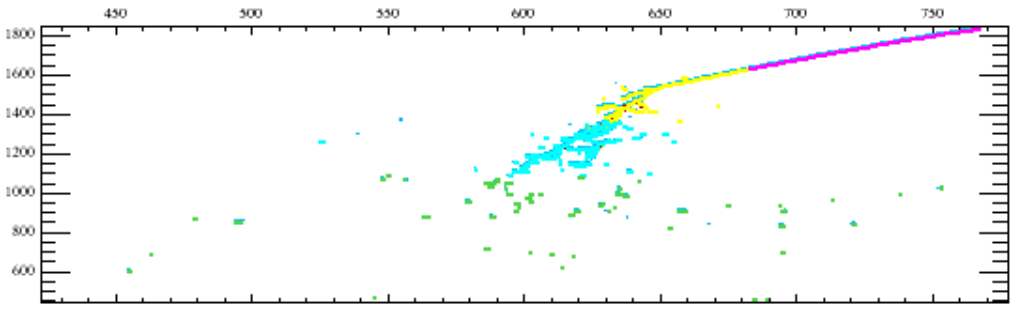
Line Merger


LArSoft
Run: 1/0
Event: 2
UTC Thu Jan 1, 19
00:00:0.01000000(
t [ticks]

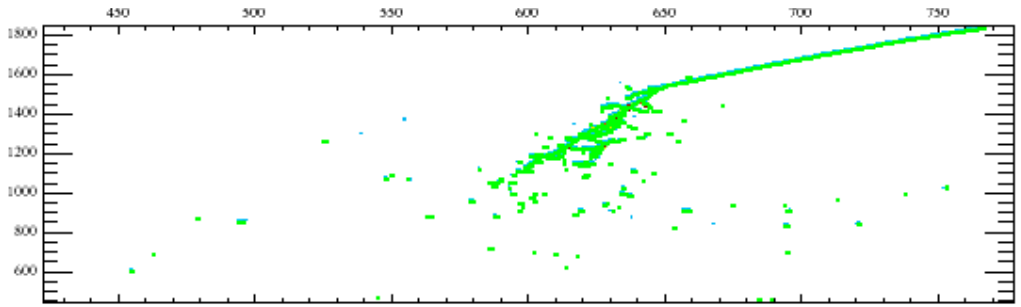

LArSoft
Run: 1/0
Event: 3
UTC Thu Jan 1, 19
00:00:0.01500000(
t [ticks]

Back up

Start with fuzzy clustering



Merge the clusters



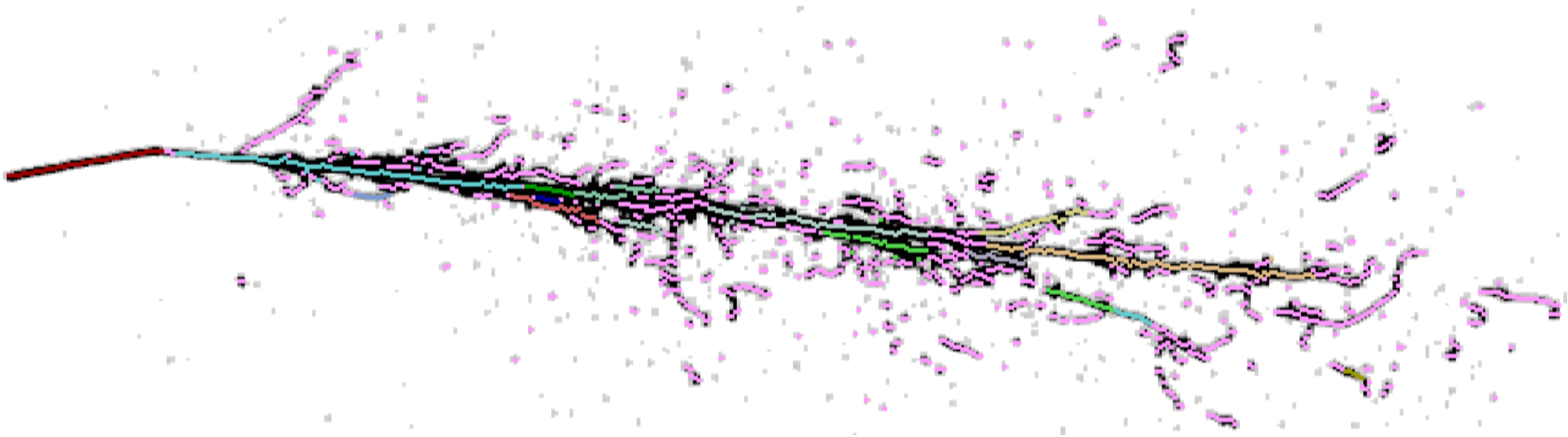
Fuzzy clustering assigns degrees of belonging, instead of definite belonging

Number of clusters determined using an optimization criteria (Xie-Beni index)

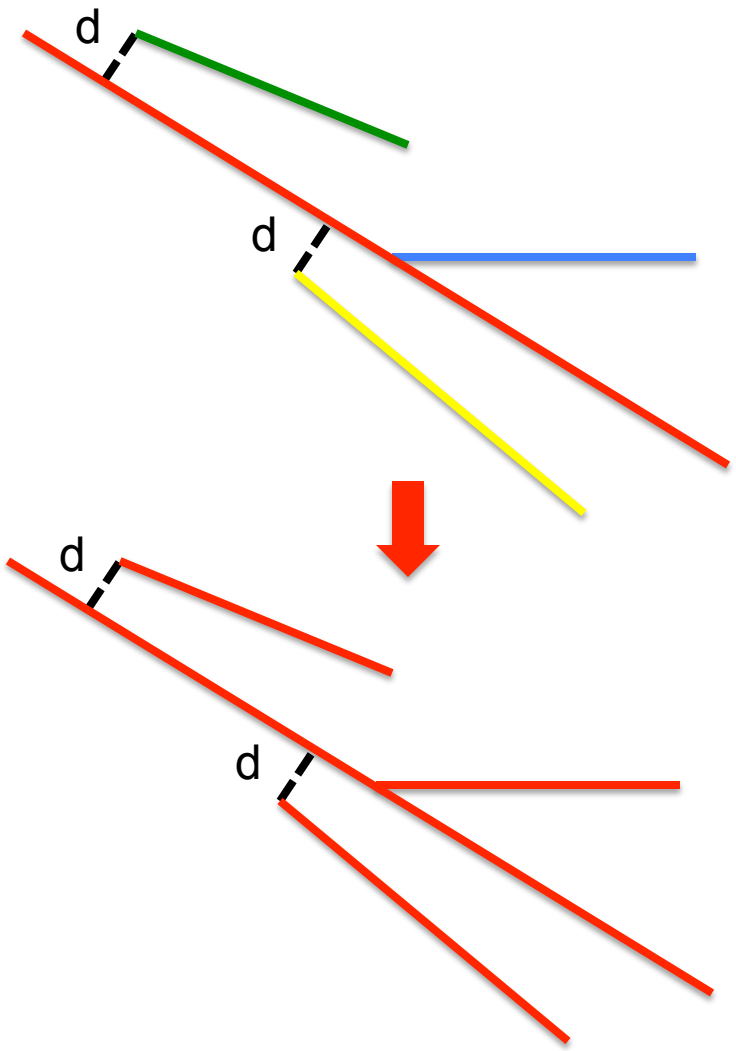
After running, the clusters are merged together

Hough line finder

- Run the Hough line finder to identify tracks and showers



Merging Hough lines

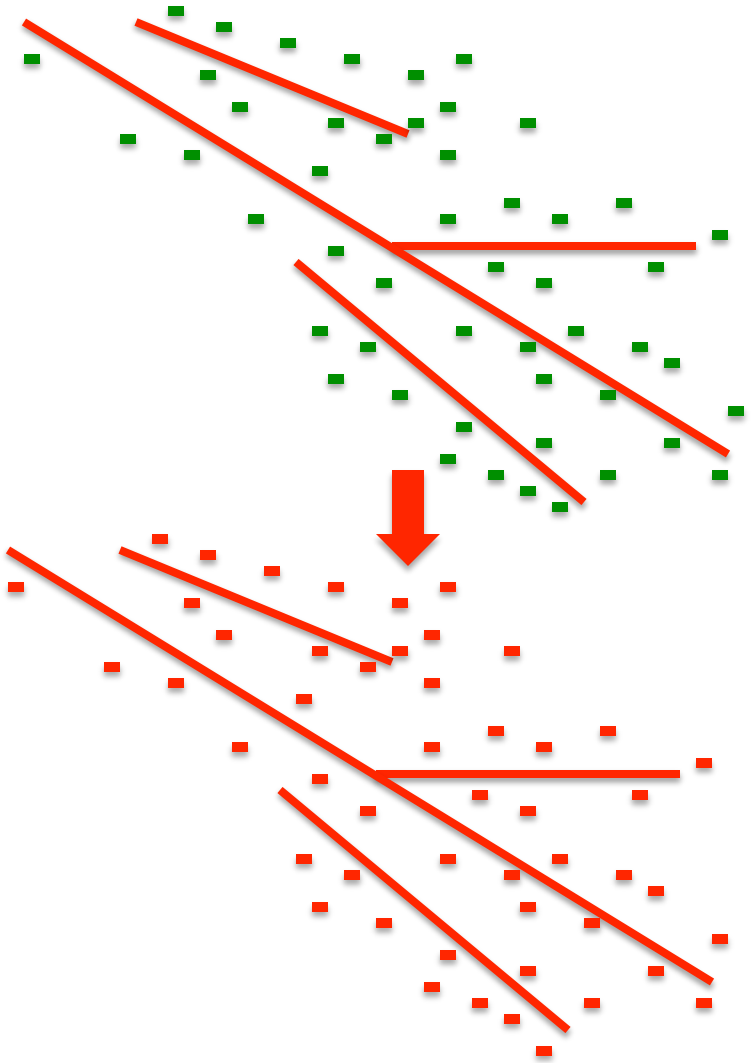


Merge Hough lines together to construct showers and tracks

Hough line finder very effective at finding lines in showers

Merge Hough lines using the distance between the line segments and angle between slopes $< 30°$

The fuzzy cluster remnants



The issue remains with what to do with the left over fuzzy cluster points

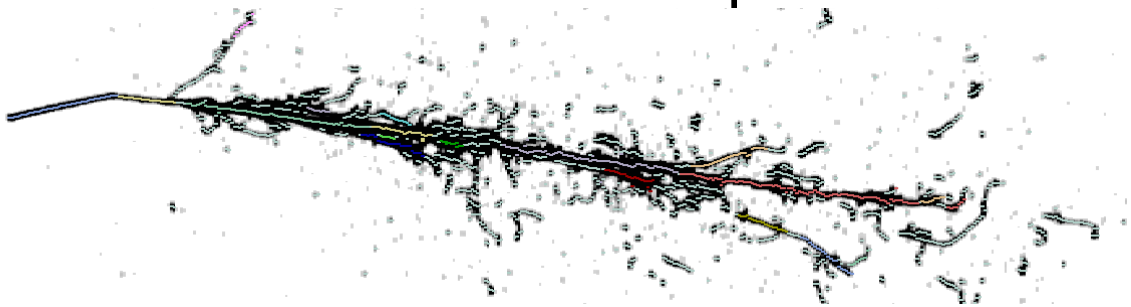
Currently merging these points into the nearest Hough line segment using:

$$d_{weighted} = \frac{distance\ point\ to\ line\ segment}{length\ of\ line\ segment}$$

Goal is to give longer line segments more weight

Status of Hough transform

- Effective at finding lines, yet very slow
- Fortunately, there are many avenues for potential improvements
- The biggest slowdown in the original HoughLineAlg resulted from refilling the accumulator from scratch after every line was found
- First optimization came from subtracting points from the original accumulator after each point was found



For this event, took processing time from 37 minutes to 6 minutes, even running at double the θ resolution

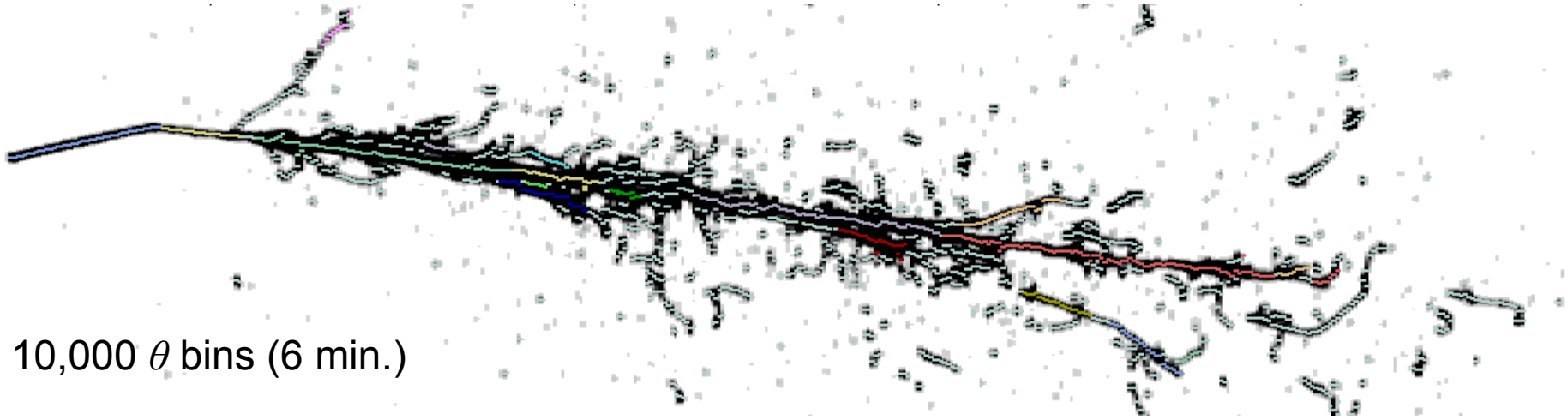
Going even faster with a PPHT

- Early work using a Progressive Probabilistic Hough Transform (PPHT) has further increased speed
- The algorithm is fairly simple:

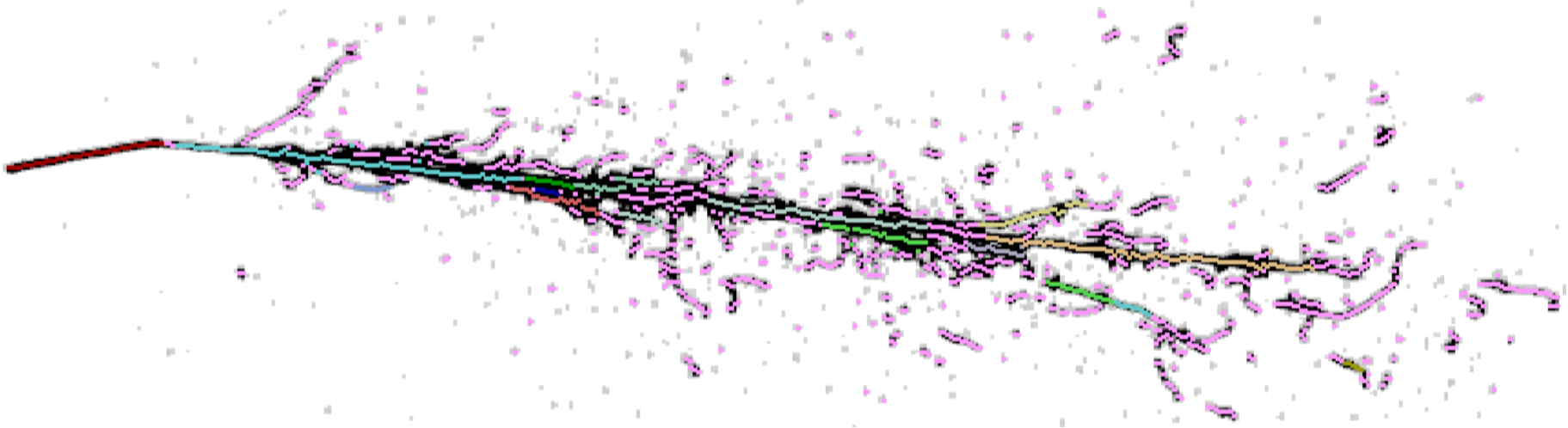
1. Randomly select one hit, remove it from the image
2. Check if the highest peak in the accumulator modified by the pixel is higher than a threshold cut; if not, go to step 1
3. Add points along the line, removing them from the image
4. Remove hits from the accumulator from the line that was found
5. Repeat step 1 if the image is not yet empty

J. Matas et al., Robust Detection of Lines Using the Progressive Probabilistic Hough Transform, Computer Vision and Image Understanding, Volume 78, Issue 1, April 2000, Pages 119–137

PPHT performance



10,000 θ bins (6 min.)



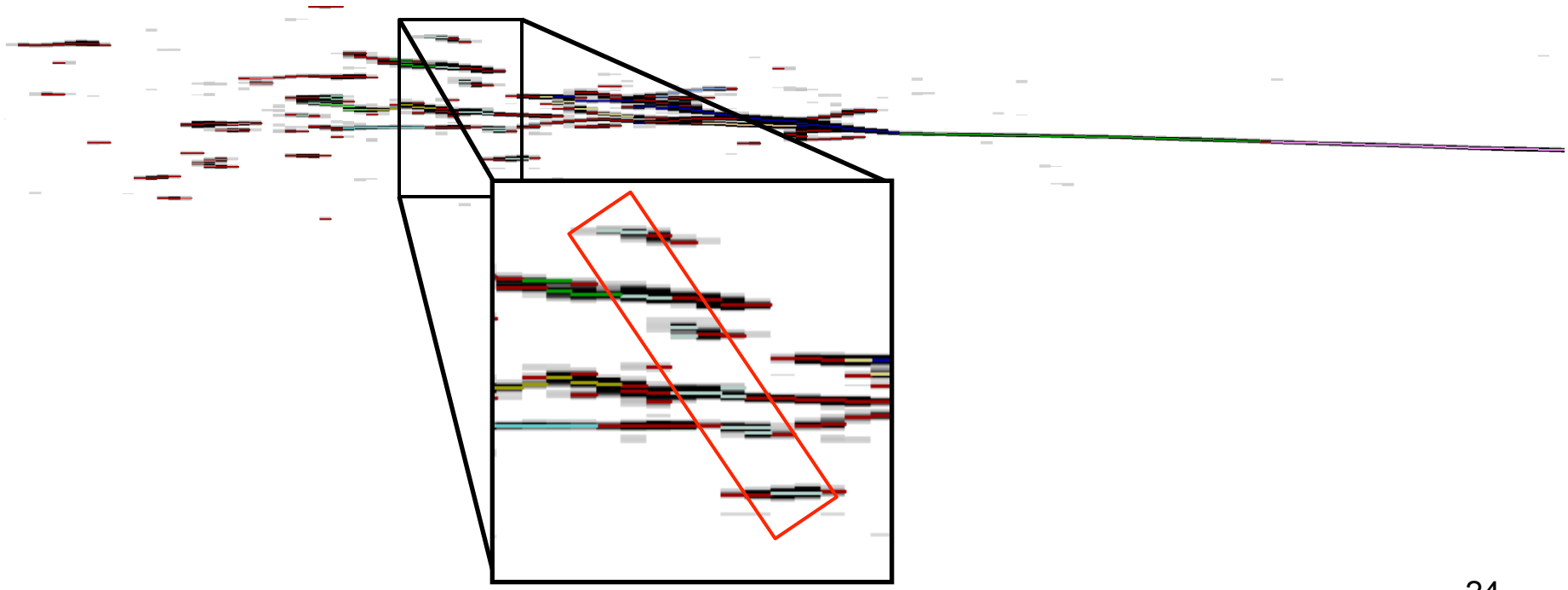
20,000 θ bins (1 min.)

Aim of the improvements

- Want to first apply a clustering algorithm (fuzzy clustering) for a rough first pass
- Then apply the PPHT to construct showers and tracks
- The PPHT is only implemented in the HoughClusAlg class, usable by fuzzy clustering code only, could be implemented in HoughLineFinder

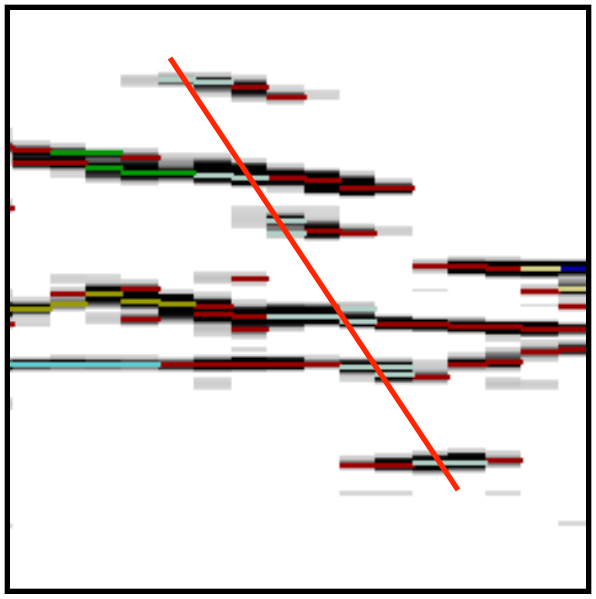
Dealing with fake lines

- The Hough line finders tend to create lines out of steep, uncorrelated hits



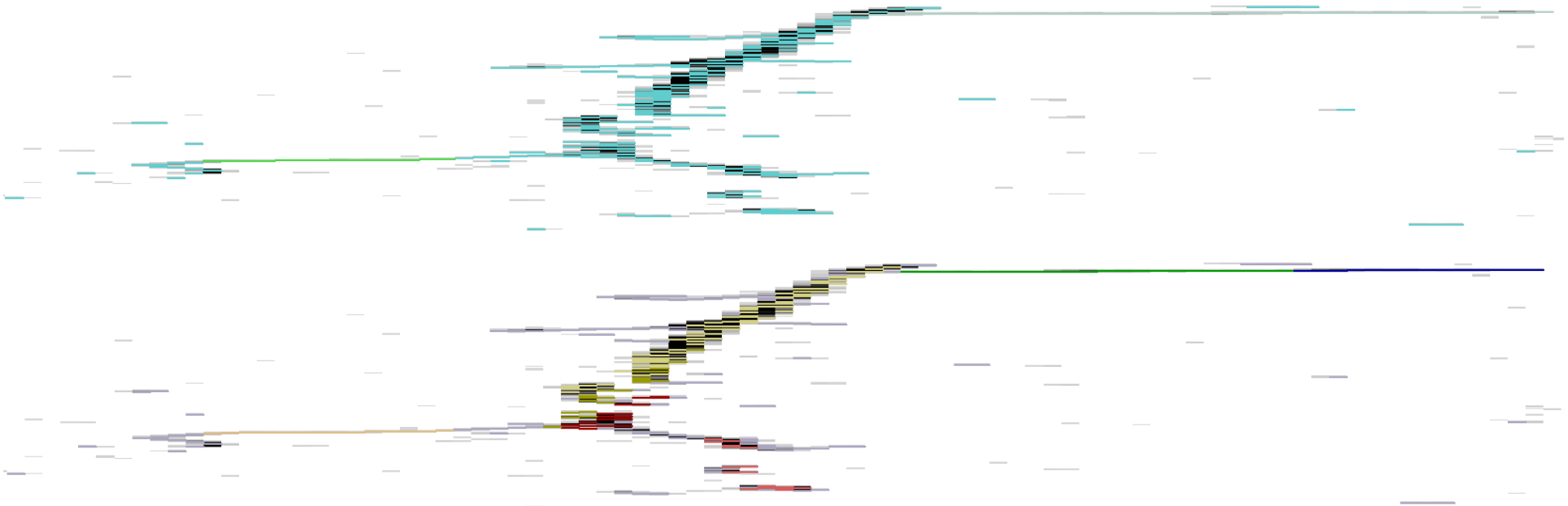
Dealing with fake lines

- Strategy walks along the line and searches for missing hits
- If too many missing hits are found, the line is vetoed



Dealing with fake lines

- Still needs some slight tuning, occasionally rejects good lines
- Most of the lines in this shower were rejected



Hough transform

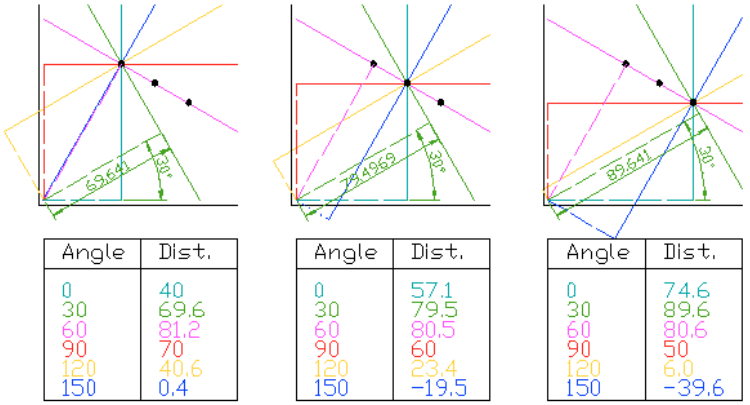
Use the parameterization:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

Fill a matrix (accumulator) in (r,θ) space with:

$$r(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta$$

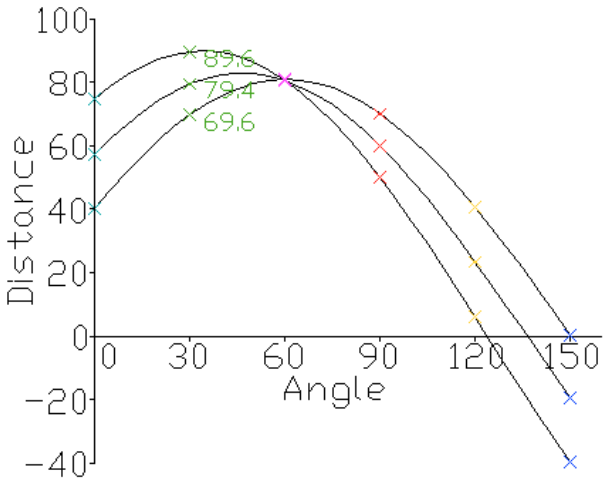
We then search the accumulator for the most populated bin, rather computationally demanding



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6



The accumulator, binned

The basic algorithm

1. *Initialize $U=[u_{ij}]$ matrix, $U^{(0)}$*
2. *At k-step: calculate the centers vectors $C^{(k)}=[c_j]$ with $U^{(k)}$*

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^m \cdot x_i}{\sum_{i=1}^{N} u_{ij}^m}$$

3. *Update $U^{(k)}$, $U^{(k+1)}$*

$$u_{ij} = \frac{1}{\sum_{k=1}^{C} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}}$$

4. *If $\| U^{(k+1)} - U^{(k)} \| < \varepsilon$ then STOP; otherwise return to step 2.*

The objective function we are trying to minimize:

$$J_m = \sum_{i=1}^{N} \sum_{j=1}^{C} u_{ij}^m \left\| x_i - c_j \right\|^2$$

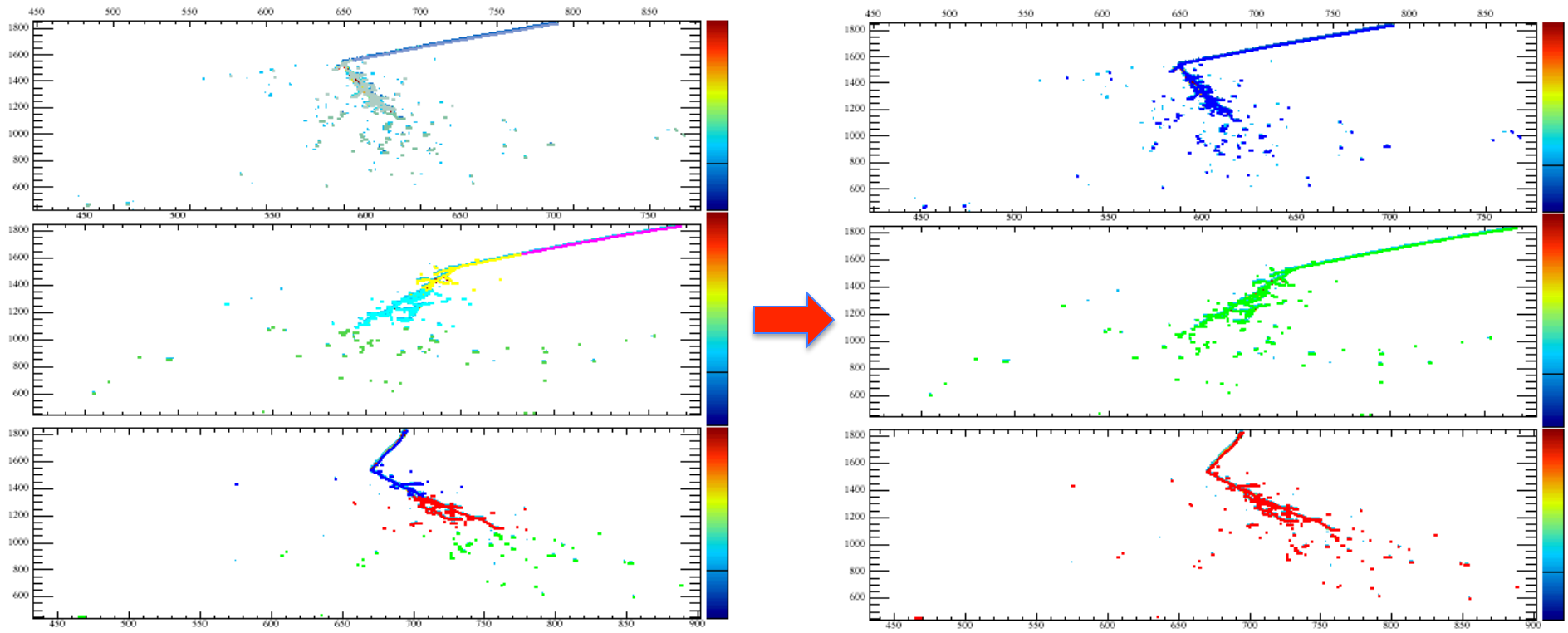
Number of clusters

- The number of clusters needs to be given as an input
- The Xie-Beni index gives us a way to evaluate how well a certain cluster number works

$$\frac{\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K} u_{ik}^{m} \left\| x_i - c_k \right\|^2}{\min_{k,\ell} \left\| c_k - c_\ell \right\|^2}$$

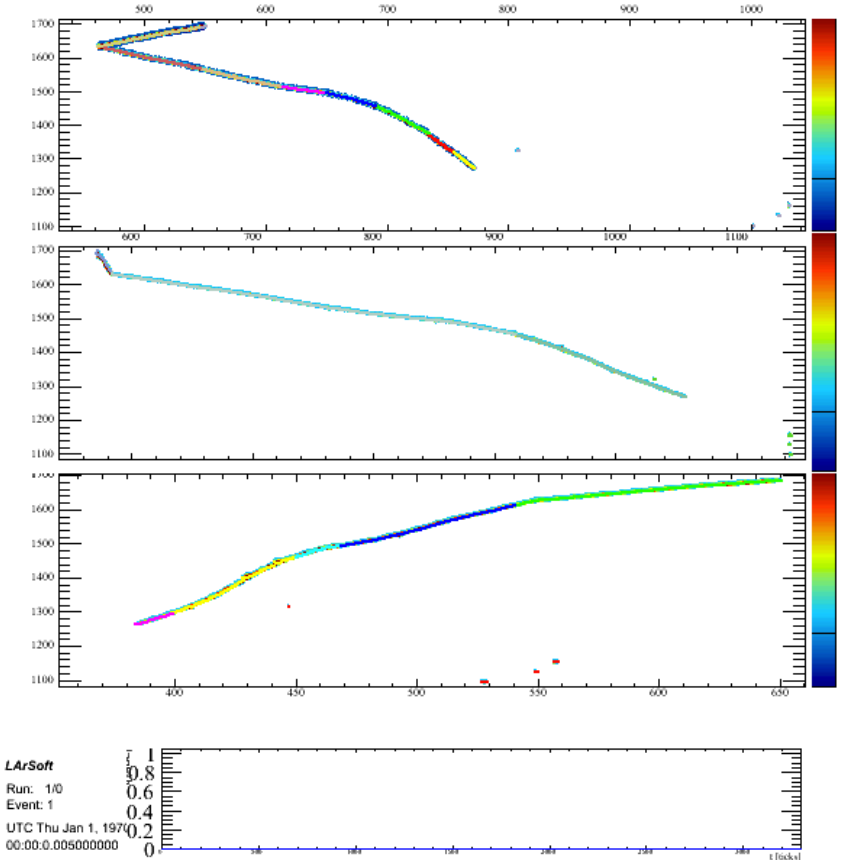
Does not work with single cluster events by definition, we'll get to that later

Look to merge clusters using Euclidean distance



1. Look for point in cluster i closest to centroid in cluster j
2. Compare that point to the point in the cluster j closest to centroid of cluster i
3. Merge the clusters

Now look for Hough lines and expand them into clusters

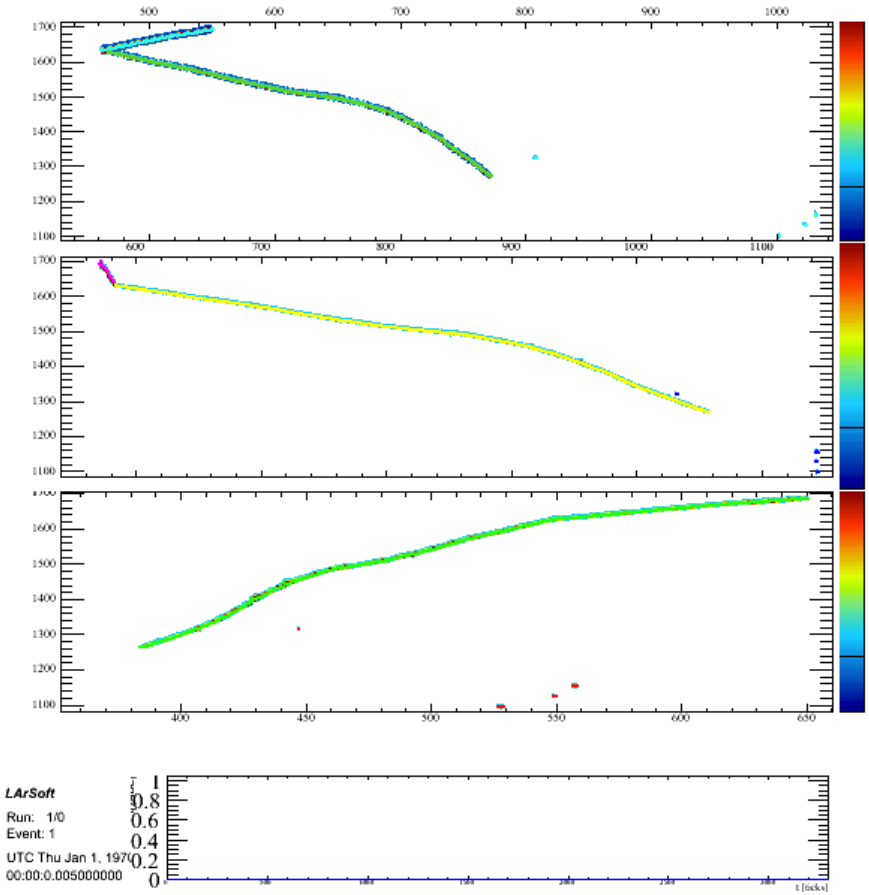


Search for Hough lines in the already identified fuzzy clusters

Hough line finder was sped up to make this practical

An issue remains with needing to merge the Hough lines though

Merging the Hough lines



Pick a line and check at its end points

If a nearby line is found, check the angle between the slopes

If the angle between the slopes is < 30°, merge the lines